Redes 2 -- Práctica 3: Añadir SSL a las comunicaciones entre el cliente y el servidor

Introducción

El objetivo de esta práctica es implementar la opción de una comunicación segura entre cliente y servidor a través de SSL.

Como compilar

Para compilar el servidor de la práctica 2 con SSL y el servidor de echo, se adjunta un fichero `G-2301-05-P3-autogen.sh` que genera los ficheros necesarios para efectuar la compilación con autotools. Después, se debe ejecutar:

```
make
```

Obtendremos dos ejecutables `./G-2301-05-P3-server` y `./G-2301-05-P3-echo`, que son el servidor de la entrega anterior con soporte SSL y el servidr de echo respectivamente.

Para compilar nuestro cliente de la práctica 1 con soporte SSL hay que estar en el directorio `cliente_SSL/` y ejecutar:

```
make -f ./G-2301-05-makefile
```

Para generar los certificados se deberá usar el siguiente comando:

```
make -f G-2301-05-makefile certificados
```

Y si fuera necesario o se quisiera borrarlos se deberá ejecutar lo siguiente:

```
make -f G-2301-05-makefile borrar_certs
```

Diseño

En esta sección están documentadas las decisiones de diseño tomadas en el desarrollo.

Módulos del programa

Los módulos de esta práctica respetan la separación de las prácticas anteriores.

Descripción y diseño

Hemos implementado las funciones que conforman la librería de SSL y para un mejor manejo hemos creado las siguientes estructuras:

```
struct Redes2_SSL_CTX {
    const SSL_METHOD*   connection_method;
    SSL_CTX*            ctx;
};

struct Redes2_SSL {
    SSL* ssl;
};
```

Estas estructuras nos permiten condensar algunas líneas de código y abstraer los distintos módulos del programa de las funciones de OpenSSL. El campo `SSL_METHOD` lo pusimos en la estructura porque el PDF decía que sería usado más de una vez, pero no ha sido nuestro caso.

La implementación de una librería que trata con OpenSSL aporta más independencia a nuestros cliente y servidor. Así si decidiéramos cambiar la implementación segura solo habría que modificar un fichero. Mucho más limpio que bucear entre líneas de código por múltiples ficheros.

En esta práctica hemos encontrado y arreglado errores de la práctica anterior. Por ejemplo nos hemos dado cuenta de que al crear un nuevo usuario lo añadíamos a la lista más de una vez. Esto afectaba a la eliminación de usuarios que tampoco funcionaba correctamente por esta razón. En el cliente de la práctica 1 también encontramos un bug que hacía que el comando `QUIT` no funcionara correctamente. Si el socket del cliente era distinto de -1 (valor que elegimos para darle al un socket aun no creado), es decir, lo que se esperaba entonces devolvíamos ERR. El arranque de esta práctica es distinto a las anteriores. Existe un `main()` en el que se crea el servidor y sus dos hilos (puerto seguro/no seguro). Tras esta operación el hilo del main muere y solo quedan en ejecución los otros dos que serán los usados por el servidor. El sistema de envío de pings lo hemos cambiado respecto a la entrega anterior porque encontramos una modo que era más sencillo y nos daba menos problemas. Antes era el servidor el encargado de mandar los pings a todos los usuarios que estuvieran en el sistema para comprobar la actividad. Sin embargo encontramos útil que fuera el propio hilo de recepción del usuario el que se mandara un ping a sí mismo. En concreto el sistema funciona de la siguiente manera. Está pensado para detectar la inactividad, es decir, mandamos un ping cuando el usuario ha estado un tiempo sin actividad, no mandamos pings cada unos cuantos segundos periódicamente. El hilo de recepción quedará bloqueado en el `recv()` como máximo 10 segundos. Si se recibe algún comando en un tiempo menor se realizarán las acciones pertinente y volverá a quedar bloqueado en `recv()`. Si no se recibe nada en 10 segundos entonces se mandará un ping al usuario, esto lo que hace es borrar una bandera del usuario que solo se restaura si este responde al servidor con un pong. Si el usuario sigue sin tener actividad y no responde al pong en otros 10 segundo se mandará otro ping y si el usuario no tiene dicha bandera será expulsado del servidor.

Hemos decidido hacer una implementación en la que convivan las conexiones seguras y no seguras. Nuestro servidor acepta conexiones simultáneamente en el puerto 6667 para conexiones no

seguras y en el 6697 para seguras. Una funcionalidad que le hemos añadido es poder cambiar el número de los puertos por los que escuchará las conexiones seguras y no seguras con una bandera. Para ejecutar esta opción se haría de la siguiente manera:

```
./G-2301-05-P3-server --port=6667 --secure=6697
```

NOTA: la documentación de las banderas esta disponible haciendo `--help`.

Conclusiones

En esta práctica hemos repasado un concepto importante en programación que es el de modularización. En concreto nos referimos a las funciones que conforman la librería de SSL. Al colocar todas las funciones encargadas de la seguridad en su propio fichero será mucho más fácil modificarlas borrarlas o en general operar con la seguridad.